

# EXHIBIT 1

**UNITED STATES DISTRICT COURT  
FOR THE EASTERN DISTRICT OF TEXAS  
MARSHALL DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

INTERNATIONAL BUSINESS  
MACHINES CORP.,

Defendant.

Case No. 2:24-cv-00064-JRG-RSP

**PLAINTIFF VIRTAMOVE, CORP.’S PRELIMINARY DISCLOSURE  
OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS**

**I. Patent Rule 3-1: Disclosure of Asserted Claims and Infringement Contentions**

Pursuant to Patent Rule 3-1, Plaintiff VirtaMove, Corp. submits the following Preliminary Disclosure of Asserted Claims and Infringement Contentions. This disclosure is based on the information available to VirtaMove as of the date of this disclosure, and VirtaMove reserves the right to amend this disclosure to the full extent permitted, consistent with the Court’s Rules and Orders.

**A. Patent Rule 3-1(a): Asserted Claims**

VirtaMove asserts that Defendant International Business Machines Corp. (“Defendant” or “IBM”) infringes the following claims (collectively, “Asserted Claims”):

- (1) U.S. Patent No. 7,519,814 (“the ’814 patent”), claims 1, 2, 6, 9, 10, and 31; and
- (2) U.S. Patent No. 7,784,058 (“the ’058 patent”), claims 1–4 and 18.

**B. Patent Rule 3-1(b): Accused Instrumentalities of which VirtaMove is aware**

VirtaMove asserts that the Asserted Claims are infringed by the various instrumentalities used, made, sold, offered for sale, or imported into the United States by Defendant, including

certain (a) IBM products and services using secure containerized applications, including without limitation IBM's Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the '814 patent; and (b) IBM products and services using user mode critical system elements as shared libraries, including without limitation IBM Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the '058 patent ("Accused Instrumentalities"). Defendant's Accused Instrumentalities of which VirtaMove is presently aware are described in more detail in the accompanying preliminary infringement contention charts.

VirtaMove reserves the right to accuse additional products from Defendant to the extent VirtaMove becomes aware of additional products during the discovery process. Unless otherwise stated, VirtaMove's assertions of infringement apply to all variations, versions, and applications of each of the Accused Instrumentalities, on information and belief, that different variations, versions, and applications of each of the Accused Instrumentalities are substantially the same for purposes of infringement of the Asserted Claims.

**C. Patent Rule 3-1(c): Claim Charts**

VirtaMove's analysis of Defendant's products is based upon limited information that is publicly available, and based on VirtaMove's own investigation prior to any discovery in these actions. Specifically, VirtaMove's analysis is based on certain limited resources that evidence certain products made, sold, used, or imported into the United States by Defendant.

VirtaMove reserves the right to amend or supplement these disclosures for any of the following reasons:

- (1) Defendant and/or third parties provide evidence relating to the Accused Instrumentalities;

- (2) VirtaMove's position on infringement of specific claims may depend on the claim constructions adopted by the Court, which has not yet occurred; and
- (3) VirtaMove's investigation and analysis of Defendant's Accused Instrumentalities is based upon public information and VirtaMove's own investigations. VirtaMove reserves the right to amend these contentions based upon discovery of non-public information that VirtaMove anticipates receiving during discovery.

Attached, and incorporated herein in their entirety, are charts identifying **where each element of the Asserted Claims are found in the Accused Instrumentalities.**

Unless otherwise indicated, the information provided that corresponds to each claim element is considered to indicate that each claim element is found within each of the different variations, versions, and applications of each of the respective Accused Instrumentalities described above.

**D. Patent Rule 3-1(d): Literal Infringement / Doctrine of Equivalents**

With respect to the patents at issue, each element of each Asserted Claim is considered to be literally present. VirtaMove also contends that each Asserted Claim is infringed or has been infringed under the doctrine of equivalents in Defendant's Accused Instrumentalities. VirtaMove also contends that Defendant both directly and indirectly infringes the Asserted Claims. For example, the Accused Instrumentalities are provided by the Defendant to customers, who are actively encouraged and instructed (for example, through Defendant's online instructions on its website and instructions, manual, or user guides that are provided with the Accused Instrumentalities) by Defendant to use the Accused Instrumentalities in ways that directly infringe the Asserted Claims. Defendant therefore specifically intends for and induces its customers to infringe the Asserted Claims under Section 271(b) through the customers' normal and customary use of the Accused Instrumentalities. In addition, Defendant is contributorily infringing the

Asserted Claims under Section 271(c) and/or Section 271(f) by selling, offering for sale, or importing the Accused Instrumentalities into the United States, which constitute a material part of the inventions claimed in the Asserted Claims, are especially made or adapted to infringe the Asserted Claims, and are otherwise not staple articles or commodities of commerce suitable for non-infringing use.

**E. Patent Rule 3-1(e): Priority Dates**

The Asserted Claims of the '814 patent are entitled to a priority date at least as early as September 15, 2003, the filing date of provisional application No. 60/502,619.

The Asserted Claims of the '058 patent are entitled to a priority date at least as early as September 22, 2003, the filing date of provisional application No. 60/504,213.

A diligent search continues for additional responsive information and VirtaMove reserves the right to supplement this response.

**F. Patent Rule 3-1(f): Identification of Instrumentalities Practicing the Claimed Invention**

At this time, VirtaMove does not identify any of its instrumentalities as practicing the Asserted Claims. A diligent search continues for additional responsive information and VirtaMove reserves the right to supplement this response.

**II. Patent Rule 3-2: Document Production Accompanying Disclosure**

Pursuant to Patent Rule 3-2, VirtaMove submits the following Document Production Accompanying Disclosure, along with an identification of the categories to which each of the documents corresponds.

**F. Patent Rule 3-2(a) documents:**

VirtaMove is presently unaware of any documents sufficient to evidence any discussion with, disclosure to, or other manner of providing to a third party, or sale of or offer to sell, the

inventions recited in the Asserted Claims of the Asserted Patents prior to the application dates or priority dates for the Asserted Patents. A diligent search continues for such documents and VirtaMove reserves the right to supplement this response.

**G. Patent Rule 3-2(b) documents:**

VirtaMove identifies the following non-privileged documents as related to evidencing conception and reduction to practice of each claimed invention of the Asserted Patents: VM\_HPE\_0000865–VM\_HPE\_0000880. A diligent search continues for additional documents and VirtaMove reserves the right to supplement this response.

**H. Patent Rule 3-2(c) documents:**

VirtaMove identifies the following documents as being the file histories for the Asserted Patents: VM\_HPE\_0000001–VM\_HPE\_0000864.

Dated: June 5, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie

CA State Bar No. 246953

Marc A. Fenster

CA State Bar No. 181067

Neil A. Rubin

CA State Bar No. 250761

Amy E. Hayden

CA State Bar No. 287026

Jacob R. Buczko

CA State Bar No. 269408

James S. Tsuei

CA State Bar No. 285530

James A. Milkey

CA State Bar No. 281283

Christian W. Conkle

CA State Bar No. 306374

Jonathan Ma

CA State Bar No. 312773

Daniel Kolko (CA SBN 341680)

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor  
Los Angeles, CA 90025  
Telephone: 310-826-7474  
Email: rmirzaie@raklaw.com  
Email: mfenster@raklaw.com  
Email: nrubin@raklaw.com  
Email: ahayden@raklaw.com  
Email: jbuczko@raklaw.com  
Email: jtsuei@raklaw.com  
Email: jmilkey@raklaw.com  
Email: cconkle@raklaw.com  
Email: jma@raklaw.com  
Email: dkolko@raklaw.com

**ATTORNEYS FOR PLAINTIFF  
VIRTAMOVE, CORP.**

**CERTIFICATE OF SERVICE**

I certify that this document is being served upon counsel of record for Defendants  
on June 5, 2024 via e-mail.

/s/ Reza Mirzaie  
Reza Mirzaie



**U.S. Patent No. 7,519,814 (“’814 Patent”)**

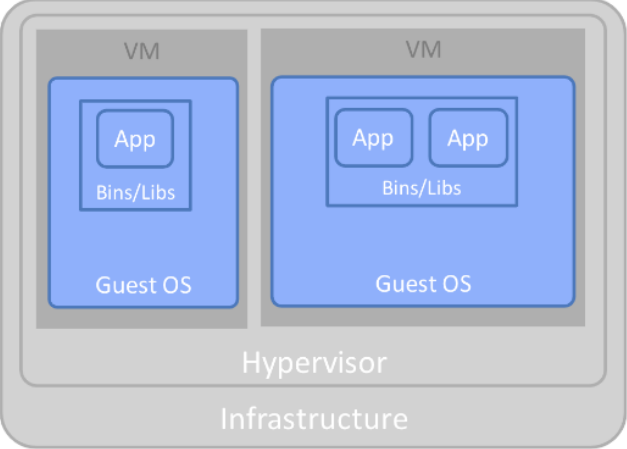
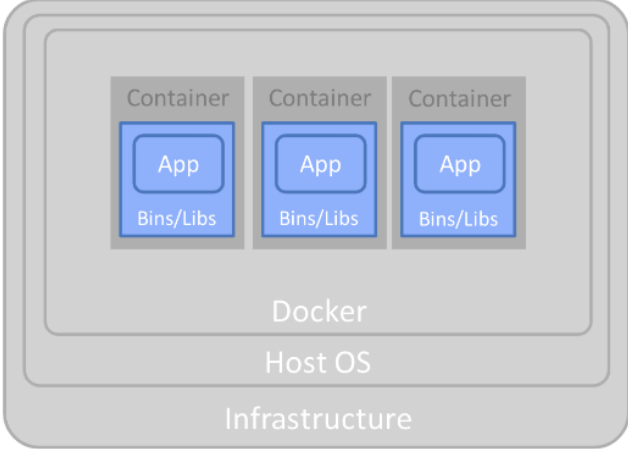
Accused Instrumentalities: IBM products and services using secure containerized applications, including without limitation IBM’s Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the asserted patent.

Each Accused Instrumentality infringes the claims in substantially the same way, and the evidence shown in this chart is similarly applicable to each Accused Instrumentality. Each claim limitation is literally infringed by each Accused Instrumentality. However, to the extent any claim limitation is not met literally, it is nonetheless met under the doctrine of equivalents because the differences between the claim limitation and each Accused Instrumentality would be insubstantial, and each Accused Instrumentality performs substantially the same function, in substantially the same way, to achieve the same result as the claimed invention. Notably, Defendant has not yet articulated which, if any, particular claim limitations it believes are not met by the Accused Instrumentalities.

**Claim 1**

Claim 1	Accused Instrumentalities
[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each	<p>To the extent the preamble is limiting, IBM practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p>For example, IBM Cloud Kubernetes Service runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p>

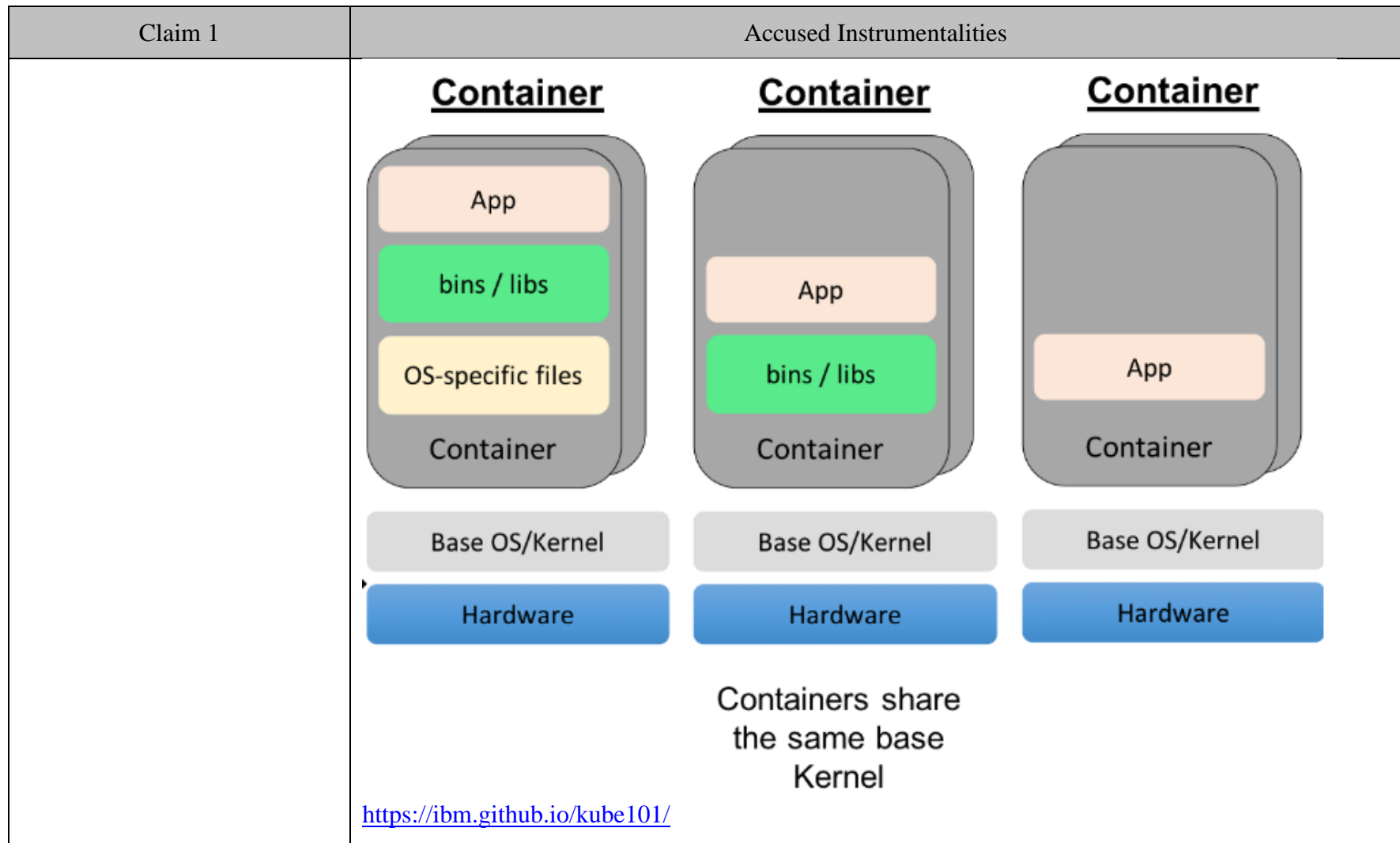
Claim 1	Accused Instrumentalities
<p>include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p><a href="https://www.ibm.com/products/kubernetes-service">https://www.ibm.com/products/kubernetes-service</a></p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p><a href="https://www.ibm.com/products/kubernetes-service">https://www.ibm.com/products/kubernetes-service</a></p> <p>Docker is an open source platform that enables developers to build, deploy, run, update and manage <i>containers</i>—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="646 289 1913 800"><div><div>Virtual Machines</div><p>The diagram illustrates the Virtual Machines architecture. It shows two separate Virtual Machines (VMs) stacked vertically. Each VM contains a Guest OS layer. Inside each Guest OS, there are one or more applications (App) and their associated binaries and libraries (Bins/Libs). The VMs are managed by a Hypervisor layer, which sits on top of the Infrastructure layer.</p></div><div><div>Containers</div><p>The diagram illustrates the Containers architecture. It shows three separate Containers stacked vertically. Each Container contains an application (App) and its associated binaries and libraries (Bins/Libs). The Containers are managed by a Docker layer, which sits on top of the Host OS layer, which in turn sits on the Infrastructure layer.</p></div></div> <p><a href="https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</a></p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p><a href="https://www.ibm.com/topics/containers">https://www.ibm.com/topics/containers</a></p> <p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"><li>• Increase development productivity by reducing the time spent on environment setup</li><li>• Eliminate issues that are caused by software dependencies</li><li>• Avoid inconsistencies when applications are run in different environments</li></ul> <p>You can use <a href="#">IBM Cloud Kubernetes Service</a> to run containers on IBM Cloud.</p> <p><a href="https://www.ibm.com/garage/method/practices/run/tool_ibm_container/">https://www.ibm.com/garage/method/practices/run/tool_ibm_container/</a>, last accessed on Nov. 17, 2023.</p>

Claim 1	Accused Instrumentalities
	<p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and disk space.</p> <p><a href="https://www.ibm.com/blog/containers-vs-vms/">https://www.ibm.com/blog/containers-vs-vms/</a></p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>



**Container**

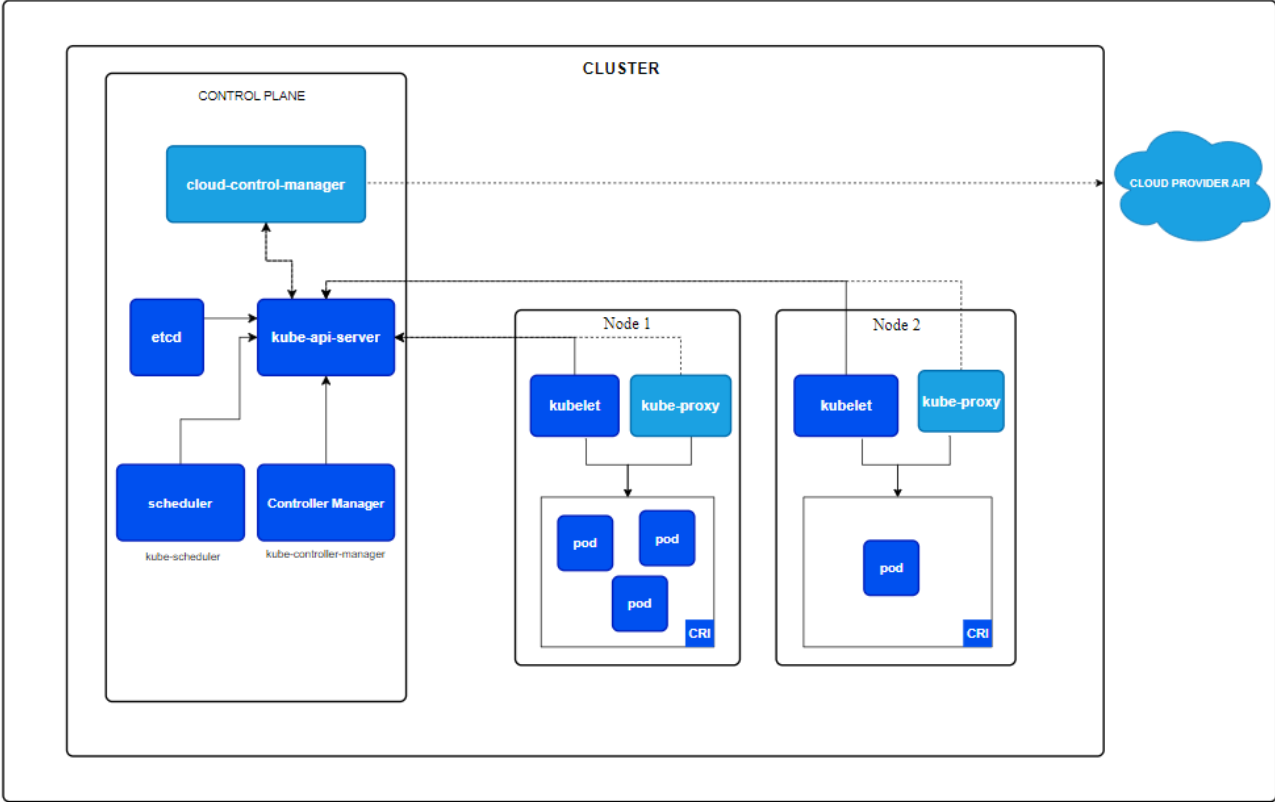


App

Container

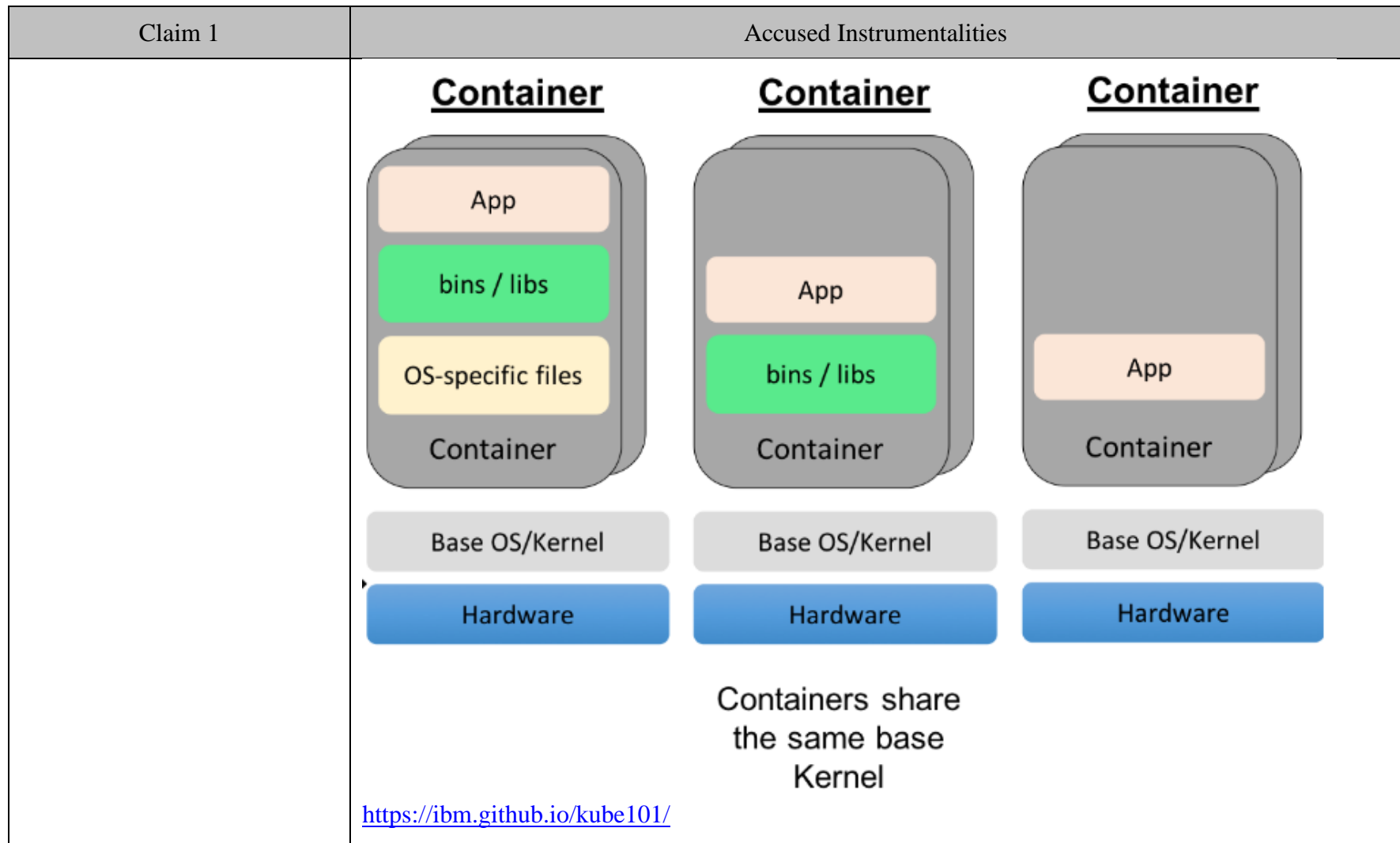
Base OS/Kernel

Hardware






Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates the Kubernetes cluster architecture. It is divided into a 'CONTROL PLANE' and 'Node 1' and 'Node 2'. The 'CONTROL PLANE' contains components: 'cloud-control-manager', 'etcd', 'kube-api-server', 'scheduler' (labeled 'kube-scheduler'), and 'Controller Manager' (labeled 'kube-controller-manager'). The 'cloud-control-manager' connects to 'etcd' and 'kube-api-server'. 'etcd' connects to 'kube-api-server'. 'scheduler' connects to 'kube-api-server'. 'Controller Manager' connects to 'kube-api-server'. 'kube-api-server' connects to 'cloud-control-manager'. 'Node 1' and 'Node 2' each contain 'kubelet' and 'kube-proxy'. 'kubelet' connects to 'kube-api-server'. 'kube-proxy' connects to 'kubelet'. Both nodes connect to a 'CLOUD PROVIDER API' cloud icon. Pods are shown running on the nodes: Node 1 has three pods, and Node 2 has one pod. A 'CRI' (Container Runtime Interface) is indicated at the bottom of each node's pod area.</p> <p>Kubernetes cluster architecture  <a href="https://kubernetes.io/docs/concepts/architecture/">https://kubernetes.io/docs/concepts/architecture/</a></p>
<p>[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container</p>	<p>The method practiced by IBM through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p>



Claim 1	Accused Instrumentalities
<p>comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>For example, IBM Cloud Kubernetes stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. Each container includes the application software as well as a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image, provided by IBM or by a third party, such as a CentOS, RHEL, or Ubuntu base image. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p><i>See, e.g.:</i></p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and disk space.</p> <p><a href="https://www.ibm.com/blog/containers-vs-vms/">https://www.ibm.com/blog/containers-vs-vms/</a></p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p>



Claim 1	Accused Instrumentalities
	<p data-bbox="667 272 982 321"><b>Container images</b></p> <p data-bbox="667 349 1266 475">A <a data-bbox="667 349 835 373" href="https://kubernetes.io/docs/concepts/containers/">container image</a> is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="634 500 1230 532"><a data-bbox="634 500 1230 532" href="https://kubernetes.io/docs/concepts/containers/">https://kubernetes.io/docs/concepts/containers/</a></p> <p data-bbox="659 581 1717 605">A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p data-bbox="659 646 1890 751">An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p data-bbox="634 824 1449 857"><a data-bbox="634 824 1449 857" href="https://cloud.ibm.com/docs/containers?topic=containers-images">https://cloud.ibm.com/docs/containers?topic=containers-images</a></p>

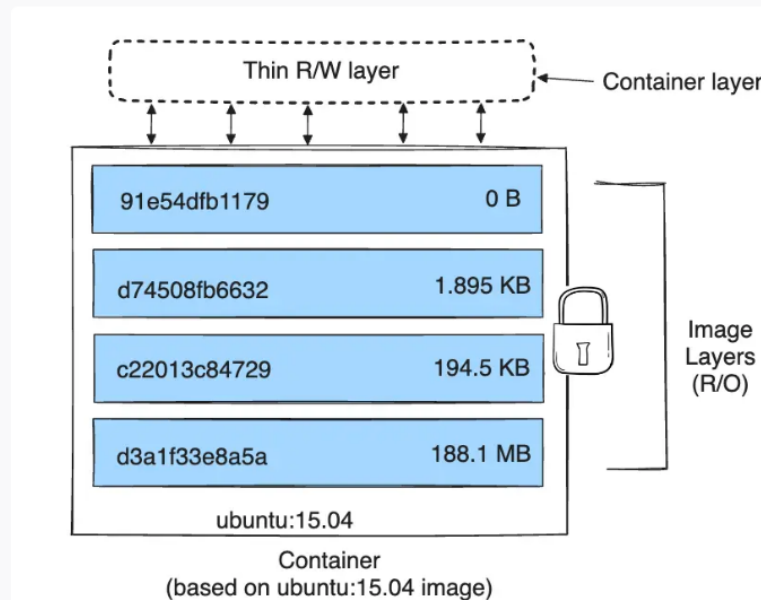
Claim 1	Accused Instrumentalities		
	<b>Docker base image</b>	<b>Supported versions</b>	<b>Source of security notices</b>
	Alpine	All stable versions with vendor security support.	<a href="#">Alpine SecDB database</a>  .
	Debian	<p>All stable versions with vendor security support.</p> <p>CVEs on binary packages that are associated with the Debian source package <code>linux</code>, such as <code>linux-libc-dev</code>, are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.</p>	<a href="#">Debian Security Bug Tracker</a>  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	<a href="#">GoogleContainerTools distroless</a>  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	<a href="#">Red Hat Security Data API</a>  .
	Ubuntu	All stable versions with vendor security support.	<a href="#">Ubuntu CVE Tracker</a>  .
	<a href="https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&amp;interface=ui">https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&amp;interface=ui</a>		

Claim 1	Accused Instrumentalities
	<h2 data-bbox="644 277 1272 342">About storage drivers</h2> <p data-bbox="644 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="644 581 1564 639">Storage drivers versus Docker volumes</h2> <p data-bbox="644 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="644 995 1902 1118">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1044 1530 1068" href="#">volumes section</a> to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1224 1182"><a data-bbox="636 1149 1224 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 272 1081 329">Images and layers</h2> <p data-bbox="657 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 513 1451 824"># syntax=docker/dockerfile:1  FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="657 889 1898 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1224 1252"><a href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 349">Volumes</h2> <p data-bbox="653 402 1906 532">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While <a href="#">bind mounts</a> are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 557 1308 589"><a href="https://kubernetes.io/docs/concepts/storage/volumes/">https://kubernetes.io/docs/concepts/storage/volumes/</a></p> <h2 data-bbox="657 641 1224 690">Container environment</h2> <p data-bbox="657 727 1474 792">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 829 1449 992" style="list-style-type: none"><li>• A filesystem, which is a combination of an <a href="#">image</a> and one or more <a href="#">volumes</a>.</li><li>• Information about the Container itself.</li><li>• Information about other objects in the cluster.</li></ul> <p data-bbox="634 1027 1528 1060"><a href="https://kubernetes.io/docs/concepts/containers/container-environment/">https://kubernetes.io/docs/concepts/containers/container-environment/</a></p>



Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 280 877 342">Images</h2> <p data-bbox="659 375 1522 526">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 565 1528 634">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 662 1329 695"><a href="https://kubernetes.io/docs/concepts/containers/images/">https://kubernetes.io/docs/concepts/containers/images/</a></p> <h2 data-bbox="653 740 919 795">Volumes</h2> <p data-bbox="653 837 1528 1279">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1307 1308 1339"><a href="https://kubernetes.io/docs/concepts/storage/volumes/">https://kubernetes.io/docs/concepts/storage/volumes/</a></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="667 284 1297 344"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="667 406 1184 453"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="667 498 1890 574"><p>This specification defines an OCI Image, consisting of an <a href="#">image manifest</a>, an <a href="#">image index</a> (optional), a set of <a href="#">filesystem layers</a>, and a <a href="#">configuration</a>.</p></div> <div data-bbox="667 609 1900 683"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="632 711 1478 781"><p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p></div>

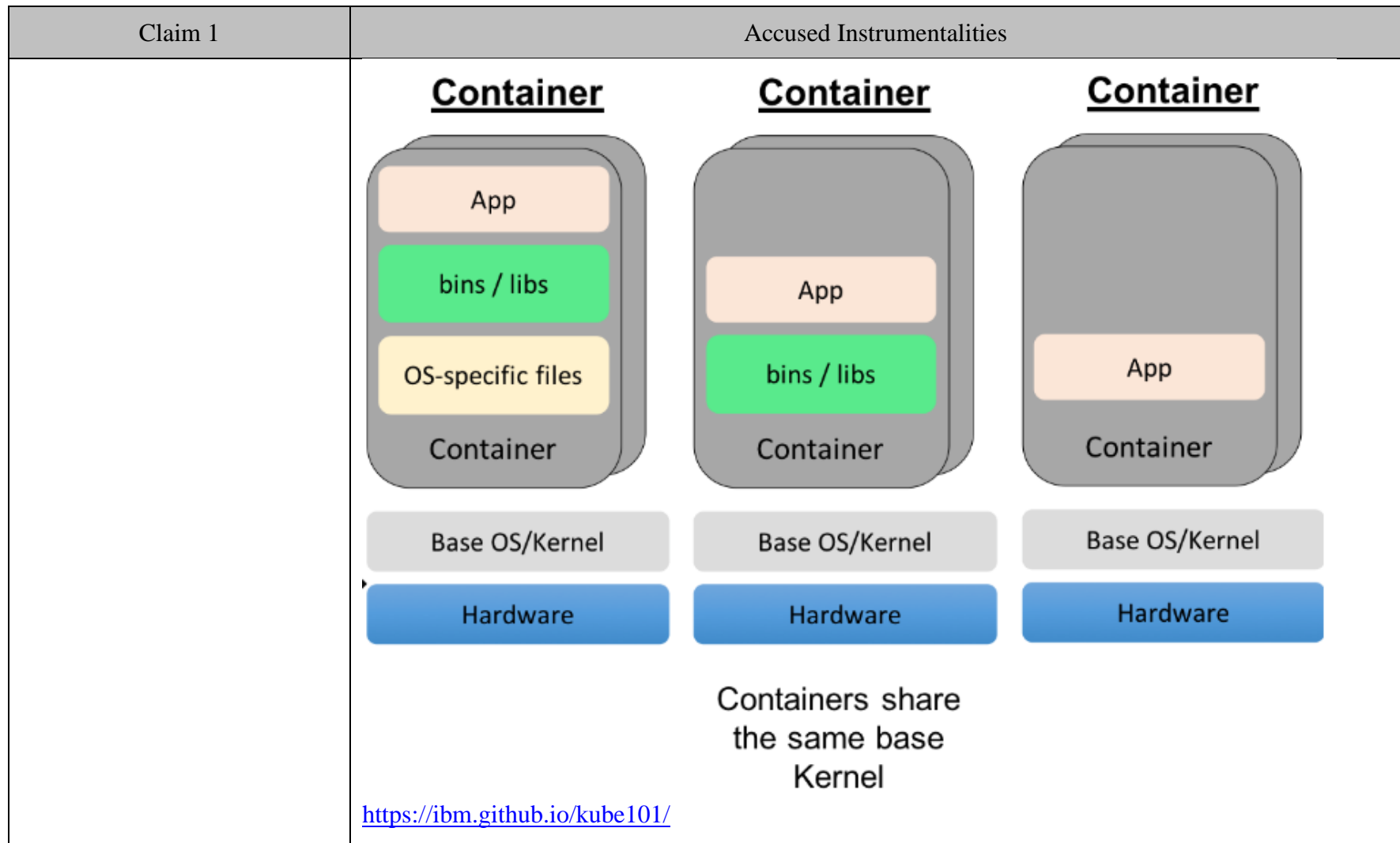
Claim 1	Accused Instrumentalities
	<p><b>Overview</b></p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more <a href="#">filesystem layer changeset</a> archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="661 649 1911 1039"> <pre> public class HelloWorld {     public static void main(String[] args) {         System.out.println("Hello, World");     } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div>   <div style="text-align: left;">/bin/java /opt/app.jar /lib/libc</div>   <p>layer</p> </div> <div style="font-size: 2em;">+</div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div>   <pre> {   "manifests": {     "platform": {       "os": "linux",       ...     }   } } </pre>   <p>image index</p> </div> <div style="font-size: 2em;">+</div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div>   <pre> {   ...   "config": {     "Cmd": [       "java", "-jar",       "app.jar"     ],     ...   } } </pre>   <p>config</p> </div> </div> </div> <p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 272 1297 334">OCI Image Configuration</h2> <p data-bbox="653 386 1913 548">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in <a href="#">Layers</a>.</p> <p data-bbox="653 586 1661 618">This section defines the <code>application/vnd.oci.image.config.v1+json</code> <a href="#">media type</a>.</p> <p data-bbox="632 651 1503 721"><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 280 747 318"><b>Layer</b></p> <ul data-bbox="688 354 1919 699" style="list-style-type: none"><li data-bbox="688 354 1241 386">• Image filesystems are composed of <i>layers</i>.</li><li data-bbox="688 402 1919 477">• Each layer represents a set of filesystem changes in a tar-based <a data-bbox="1507 402 1656 435" href="#">layer format</a>, recording files to be added, changed, or deleted relative to its parent layer.</li><li data-bbox="688 493 1919 568">• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.</li><li data-bbox="688 584 1919 699">• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem.</li></ul> <p data-bbox="661 751 856 789"><b>Image JSON</b></p> <ul data-bbox="688 824 1919 1170" style="list-style-type: none"><li data-bbox="688 824 1919 940">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.</li><li data-bbox="688 956 1919 1031">• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.</li><li data-bbox="688 1047 1919 1122">• This JSON is considered to be immutable, because changing it would change the computed <a data-bbox="720 1092 827 1125" href="#">ImageID</a>.</li><li data-bbox="688 1138 1822 1170">• Changing it means creating a new derived image, instead of changing the existing image.</li></ul> <p data-bbox="634 1203 1503 1269"><a data-bbox="634 1203 1503 1269" href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>






Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> <li>• <b>rootfs object</b>, REQUIRED</li> </ul> <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> <li>◦ <b>type string</b>, REQUIRED</li> </ul> <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> <li>◦ <b>diff_ids array of strings</b>, REQUIRED</li> </ul> <p>An array of layer content hashes ( <code>DiffIDs</code> ), in order from first to last.</p> <p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>
<p>[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p>The system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>





Claim 1	Accused Instrumentalities
	<p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p><a href="https://cloud.ibm.com/docs/containers?topic=containers-images">https://cloud.ibm.com/docs/containers?topic=containers-images</a></p>

Claim 1	Accused Instrumentalities		
<div></div>	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	<a href="#">Alpine SecDB database</a>  .
	Debian	<p>All stable versions with vendor security support.</p> <p>CVEs on binary packages that are associated with the Debian source package <code>linux</code>, such as <code>linux-libc-dev</code>, are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.</p>	<a href="#">Debian Security Bug Tracker</a>  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	<a href="#">GoogleContainerTools distroless</a>  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	<a href="#">Red Hat Security Data API</a>  .
	Ubuntu	All stable versions with vendor security support.	<a href="#">Ubuntu CVE Tracker</a>  .
	<a href="https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&amp;interface=ui">https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&amp;interface=ui</a>		
[1c] the containers of application software excluding a kernel,	<p>In the method practiced by IBM through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p>		

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="640 272 1843 1182"> <p style="text-align: center;">Containers share the same base Kernel</p> <p><a href="https://ibm.github.io/kube101/">https://ibm.github.io/kube101/</a></p> </div>
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the	In the method practiced by IBM through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.

Claim 1	Accused Instrumentalities
<p>associated local system files that remain resident on the server,</p>	<p>For example, each container will utilize its own local system files, including libraries such as libc/glibc and configuration files, not the corresponding libraries and configuration files of the host OS.</p> <p><i>See, e.g.:</i></p> <p>Rather than spinning up an entire virtual machine, <a href="#">containerization</a> packages together everything needed to run a single application or microservice (along with runtime libraries they need to run). The container includes all the code, its dependencies and even the operating system itself. This enables applications to run almost anywhere — a desktop computer, a traditional IT infrastructure or the cloud.</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p><a href="https://www.ibm.com/blog/containers-vs-vms/">https://www.ibm.com/blog/containers-vs-vms/</a></p>
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than <a href="#">virtual machines (VMs)</a>, containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>

Claim 1	Accused Instrumentalities
	<p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> <li>• Increase development productivity by reducing the time spent on environment setup</li> <li>• Eliminate issues that are caused by software dependencies</li> <li>• Avoid inconsistencies when applications are run in different environments</li> </ul> <p>You can use <a href="#">IBM Cloud Kubernetes Service</a> to run containers on IBM Cloud.</p> <p><a href="https://www.ibm.com/garage/method/practices/run/tool_ibm_container/">https://www.ibm.com/garage/method/practices/run/tool_ibm_container/</a>, last accessed on Nov. 17, 2023.</p>
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.</p> <p><i>See, e.g.:</i></p> <p><a href="#">Containers</a> are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple <a href="#">virtual machines (VMs)</a> to share the CPU, memory and other resources of a single hardware server.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p>

Claim 1	Accused Instrumentalities
	<p><b>Fault isolation:</b> Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>

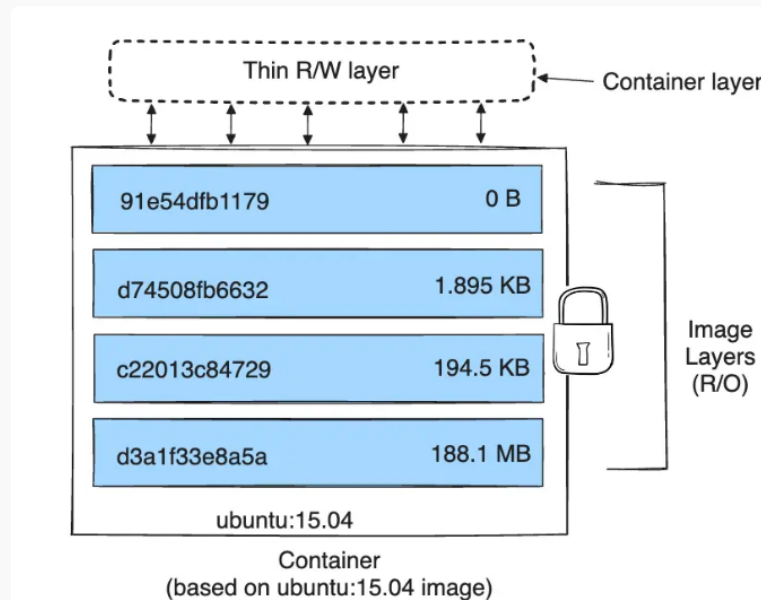


Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 277 1272 342">About storage drivers</h2> <p data-bbox="646 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 581 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 992 1902 1117">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1040 1535 1068" href="#">volumes section</a> to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1224 1182"><a data-bbox="636 1149 1224 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 272 1081 329">Images and layers</h2> <p data-bbox="657 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 513 1451 824"># syntax=docker/dockerfile:1  FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="657 889 1900 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1224 1252"><a href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

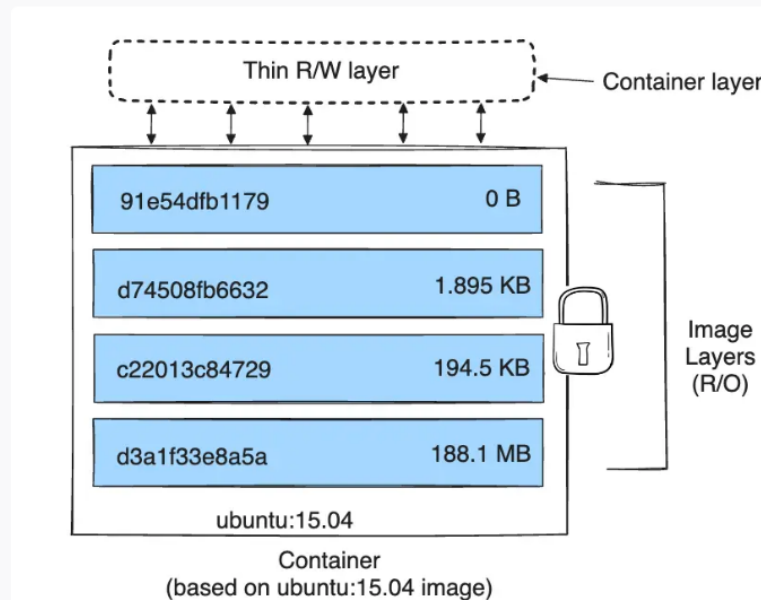
Claim 1	Accused Instrumentalities
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Limit the number of privileged containers. Containers run as a separate Linux process on the compute host that is isolated from other processes. Although users have root access inside the container, the permissions of this user are limited outside the container to protect other Linux processes, the host file system, and host devices. Some apps require access to the host file system or advanced permissions to run properly. You can run containers in privileged mode to allow the container the same access as the processes running on the compute host. Keep in mind that privileged containers can cause huge damage to the cluster and the underlying compute host if they become compromised. Try to limit the number of containers that run in privileged mode and consider changing the configuration for your app so that the app can run without advanced permissions.</p> <p><a href="https://cloud.ibm.com/docs/containers?topic=containers-security">https://cloud.ibm.com/docs/containers?topic=containers-security</a></p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple <i>virtual machines</i> (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 277 1272 342">About storage drivers</h2> <p data-bbox="646 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 581 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 992 1902 1117">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1040 1535 1068" href="#">volumes section</a> to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1224 1182"><a data-bbox="636 1149 1224 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 272 1081 329">Images and layers</h2> <p data-bbox="657 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 513 1451 824"># syntax=docker/dockerfile:1  FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="657 889 1898 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1224 1252"><a href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</a></p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 349">Volumes</h2> <p data-bbox="653 402 1906 532">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While <a href="#">bind mounts</a> are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 557 1308 589"><a href="https://kubernetes.io/docs/concepts/storage/volumes/">https://kubernetes.io/docs/concepts/storage/volumes/</a></p> <h2 data-bbox="653 638 1226 690">Container environment</h2> <p data-bbox="653 727 1474 792">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 829 1451 992" style="list-style-type: none"><li>• A filesystem, which is a combination of an <a href="#">image</a> and one or more <a href="#">volumes</a>.</li><li>• Information about the Container itself.</li><li>• Information about other objects in the cluster.</li></ul> <p data-bbox="634 1027 1528 1060"><a href="https://kubernetes.io/docs/concepts/containers/container-environment/">https://kubernetes.io/docs/concepts/containers/container-environment/</a></p>



Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 280 877 342">Images</h2> <p data-bbox="659 375 1522 526">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 565 1528 634">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="634 662 1329 695"><a href="https://kubernetes.io/docs/concepts/containers/images/">https://kubernetes.io/docs/concepts/containers/images/</a></p> <h2 data-bbox="653 740 919 795">Volumes</h2> <p data-bbox="653 837 1528 1279">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="634 1307 1308 1339"><a href="https://kubernetes.io/docs/concepts/storage/volumes/">https://kubernetes.io/docs/concepts/storage/volumes/</a></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="667 284 1297 344"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="667 406 1184 453"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="667 498 1892 574"><p>This specification defines an OCI Image, consisting of an <a href="#">image manifest</a>, an <a href="#">image index</a> (optional), a set of <a href="#">filesystem layers</a>, and a <a href="#">configuration</a>.</p></div> <div data-bbox="667 609 1900 683"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="632 711 1478 781"><p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p></div>

Claim 1	Accused Instrumentalities
	<p><b>Overview</b></p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more <a href="#">filesystem layer changeset</a> archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="661 649 1911 1039"> <pre> public class HelloWorld {     public static void main(String[] args) {         System.out.println("Hello, World");     } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 100px; height: 100px; margin: 10px auto; display: flex; align-items: center; justify-content: center;"> <div style="text-align: left; padding: 5px;"> /bin/java  /opt/app.jar  /lib/libc </div> </div> <p>layer</p> </div> <div style="font-size: 2em;">+</div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: 150px;"> <pre> {   "manifests": {     "platform": {       "os": "linux",       ...     }   } } </pre> </div> <p>image index</p> </div> <div style="font-size: 2em;">+</div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: 150px;"> <pre> {   ...   "config": {     "Cmd": [       "java", "-jar",       "app.jar"     ],     ...   } } </pre> </div> <p>config</p> </div> </div> </div> <p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 272 1297 334">OCI Image Configuration</h2> <p data-bbox="653 386 1913 548">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in <a href="#">Layers</a>.</p> <p data-bbox="653 586 1661 618">This section defines the <code>application/vnd.oci.image.config.v1+json</code> <a href="#">media type</a>.</p> <p data-bbox="632 651 1503 721"><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 280 747 318"><b>Layer</b></p> <ul data-bbox="688 354 1919 699" style="list-style-type: none"><li data-bbox="688 354 1241 386">• Image filesystems are composed of <i>layers</i>.</li><li data-bbox="688 402 1919 477">• Each layer represents a set of filesystem changes in a tar-based <a data-bbox="1503 402 1656 435" href="#">layer format</a>, recording files to be added, changed, or deleted relative to its parent layer.</li><li data-bbox="688 493 1919 568">• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.</li><li data-bbox="688 584 1919 699">• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem.</li></ul> <p data-bbox="661 751 856 789"><b>Image JSON</b></p> <ul data-bbox="688 824 1919 1170" style="list-style-type: none"><li data-bbox="688 824 1919 940">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.</li><li data-bbox="688 956 1919 1031">• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.</li><li data-bbox="688 1047 1919 1122">• This JSON is considered to be immutable, because changing it would change the computed <a data-bbox="718 1092 827 1125" href="#">ImageID</a>.</li><li data-bbox="688 1138 1822 1170">• Changing it means creating a new derived image, instead of changing the existing image.</li></ul> <p data-bbox="634 1203 1503 1269"><a data-bbox="634 1203 1503 1269" href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> <li>• <b>rootfs object</b>, REQUIRED</li> </ul> <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> <li>◦ <b>type string</b>, REQUIRED</li> </ul> <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> <li>◦ <b>diff_ids array of strings</b>, REQUIRED</li> </ul> <p>An array of layer content hashes ( <code>DiffIDs</code> ), in order from first to last.</p> <p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>

## Claim 2

Claim 2	Accused Instrumentalities
2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.</p> <p>For example, a container image has an associated image configuration comprising information for starting the one or more applications. This can be an Open Containers Initiative image configuration.</p> <p><i>See, e.g.:</i></p>

Claim 2	Accused Instrumentalities
	<div data-bbox="625 256 1251 315"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="625 378 1136 423"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="625 470 1848 545"><p>This specification defines an OCI Image, consisting of an <a href="#">image manifest</a>, an <a href="#">image index</a> (optional), a set of <a href="#">filesystem layers</a>, and a <a href="#">configuration</a>.</p></div> <div data-bbox="625 579 1854 654"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="588 683 1432 751"><p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p></div>

Claim 2	Accused Instrumentalities
	<p data-bbox="604 250 785 289"><b>Overview</b></p> <p data-bbox="604 345 1848 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more <a href="#">filesystem layer changeset</a> archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="625 630 1864 1008"> <p>The diagram illustrates the components of an OCI image. On the left, a code block for a 'HelloWorld' class is shown. An arrow points from this code to a cylinder labeled 'layer', which contains the paths '/bin/java', '/opt/app.jar', and '/lib/libc'. To the right of the layer is a plus sign, followed by a document icon labeled 'image index' containing a JSON snippet for 'manifests'. Another plus sign follows, leading to a document icon labeled 'config' containing a JSON snippet for 'config' with a 'Cmd' array.</p> <pre> public class HelloWorld {     public static void main(String[] args) {         System.out.println("Hello, World");     } } </pre> <p>→</p> <p>/bin/java /opt/app.jar /lib/libc</p> <p>+</p> <pre> {   "manifests": {     "platform": {       "os": "linux",       ...     }   } } </pre> <p>+</p> <pre> {   ...   "config": {     "Cmd": [       "java", "-jar",       "app.jar"     ],     ...   } } </pre> <p>layer                      image index                      config</p> <p data-bbox="588 1036 1432 1105"><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</a></p> </div>



Claim 2	Accused Instrumentalities
	<h2 data-bbox="604 245 1251 305">OCI Image Configuration</h2> <p data-bbox="604 358 1871 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in <a href="#">Layers</a>.</p> <p data-bbox="604 558 1612 591">This section defines the <code>application/vnd.oci.image.config.v1+json</code> <a href="#">media type</a>.</p> <p data-bbox="588 623 1457 695"><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p>

Claim 2	Accused Instrumentalities
	<ul style="list-style-type: none"><li>• <b>config object</b>, OPTIONAL<p>The execution parameters which SHOULD be used as a base when running a container using the image. This field can be <code>null</code> , in which case any execution parameters should be specified at creation of the container.</p><ul style="list-style-type: none"><li>◦ <b>Env array of strings</b>, OPTIONAL<p>Entries are in the format of <code>VARNAME=VARVALUE</code> . These values act as defaults and are merged with any specified when creating a container.</p></li><li>◦ <b>Entrypoint array of strings</b>, OPTIONAL<p>A list of arguments to use as the command to execute when the container starts. These values act as defaults and may be replaced by an entrypoint specified when creating a container.</p></li><li>◦ <b>Cmd array of strings</b>, OPTIONAL<p>Default arguments to the entrypoint of the container. These values act as defaults and may be replaced by any specified when creating a container. If an <code>Entrypoint</code> value is not specified, then the first entry of the <code>Cmd</code> array SHOULD be interpreted as the executable to run.</p><p><a href="https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</a></p></li></ul></li></ul>

**Claim 6**

Claim 6	Accused Instrumentalities
6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.</p> <p>For example, Kubernetes containers have an associated hostname, which in the case of a single-container Pod is the unique identity of that container. For another example, Kubernetes pods have an associated hostname, which is unique. For another example, a networked Kubernetes pod has an assigned IPv4 and/or IPv6 address. For another example, a Docker container has an IP address and a hostname.</p> <p><i>See, e.g.:</i></p> <h3>Container information</h3> <p>The <i>hostname</i> of a Container is the name of the Pod in which the Container is running. It is available through the <code>hostname</code> command or the <code>gethostname</code> function call in libc.</p> <p>The Pod name and namespace are available as environment variables through the <a href="#">downward API</a>.</p> <p>User defined environment variables from the Pod definition are also available to the Container, as are any environment variables specified statically in the container image.</p> <p><a href="https://kubernetes.io/docs/concepts/containers/container-environment/">https://kubernetes.io/docs/concepts/containers/container-environment/</a></p>

Claim 6	Accused Instrumentalities
	<p data-bbox="604 245 1203 293"><b>IP address and hostname</b></p> <p data-bbox="604 337 1864 461">By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.</p> <p data-bbox="604 511 1843 683">You can connect a running container to multiple networks, either by passing the <code>--network</code> flag multiple times when creating the container, or using the <code>docker network connect</code> command for already running containers. In both cases, you can use the <code>--ip</code> or <code>--ip6</code> flags to specify the container's IP address on that particular network.</p> <p data-bbox="604 734 1850 857">In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using <code>--hostname</code>. When connecting to an existing network using <code>docker network connect</code>, you can use the <code>--alias</code> flag to specify an additional network alias for the container on that network.</p> <p data-bbox="588 889 1016 919"><a href="https://docs.docker.com/network/">https://docs.docker.com/network/</a></p>

### Claim 9

Claim 9	Accused Instrumentalities
<p data-bbox="203 1078 684 1365">9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p>	<p data-bbox="714 1078 1850 1219">IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p> <p data-bbox="714 1247 1843 1354">For example, Kubernetes tracks and limits resource usage, including CPU and memory resources. For another example, Docker tracks and limits resource usage, including CPU and memory resources.</p> <p data-bbox="714 1377 831 1409"><i>See, e.g.:</i></p>

### **Resource Management for Pods and Containers**

When you specify a Pod, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM); there are others.

When you specify the resource *request* for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on. When you specify a resource *limit* for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set. The kubelet also reserves at least the *request* amount of that system resource specifically for that container to use.

#### Requests and limits

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its *request* for that resource specifies. However, a container is not allowed to use more than its resource *limit*.

For example, if you set a *memory request* of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.

If you set a *memory limit* of 4GiB for that container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than

Claim 9	Accused Instrumentalities
	<p>the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.</p> <p>Limits can be implemented either reactively (the system intervenes once it sees a violation) or by enforcement (the system prevents the container from ever exceeding the limit). Different runtimes can have different ways to implement the same restrictions.</p> <p><a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a></p> <p><b>Runtime options with Memory, CPUs, and GPUs</b></p> <p>By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags of the <code>docker run</code> command. This section provides details on when you should set such limits and the possible implications of setting them.</p> <p><b>Limit a container's access to memory</b></p> <p>Docker can enforce hard or soft memory limits.</p> <ul style="list-style-type: none"><li>• Hard limits lets the container use no more than a fixed amount of memory.</li><li>• Soft limits lets the container use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine.</li></ul> <p><a href="https://docs.docker.com/config/containers/resource_constraints/">https://docs.docker.com/config/containers/resource_constraints/</a></p>

### Claim 10

Claim 10	Accused Instrumentalities
10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.</p> <p><i>See, e.g.:</i></p> <p><a href="#">Containers</a> are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple <a href="#">virtual machines (VMs)</a> to share the CPU, memory and other resources of a single hardware server.</p> <p><a href="https://www.ibm.com/topics/docker">https://www.ibm.com/topics/docker</a></p> <p><b>Fault isolation:</b> Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p><a href="https://www.ibm.com/topics/containerization">https://www.ibm.com/topics/containerization</a></p>

### Claim 31

Claim 31	Accused Instrumentalities
[31pre] A computing system for performing a plurality of tasks each	To the extent the preamble is construed as a limitation, each Accused Instrumentality is or comprises a computing system for performing a plurality of tasks each comprising a plurality of processes.


Claim 31	Accused Instrumentalities
comprising a plurality of processes comprising:	<i>See</i> claim limitations below. <i>See also</i> analysis and evidence for [1pre] above.
[31a] a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address	<p>Each Accused Instrumentality comprises a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address.</p> <p><i>See</i> analysis and evidence for [1pre], limitations [1a] and [1f], and claim 6 above.</p>
[31b] wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively	<p>Each Accused Instrumentality comprises a system wherein the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel.</p> <p><i>See</i> analysis and evidence for [1pre], limitations [1a], [1c], [1d], [1e], and [1f], and claim 2 above.</p>



Claim 31	Accused Instrumentalities
uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,	
[31c] a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.	<p>Each Accused Instrumentality comprises a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.</p> <p>For example, IBM Cloud Kubernetes Service includes the containerd runtime module or another container runtime. For another example, Kubernetes uses the Linux kernel's seccomp mode to monitor and control system calls made from a container.</p> <p><i>See, e.g.:</i></p> <p>Security Bulletin: IBM Cloud Kubernetes Service is affected by a containerd security vulnerability (CVE-2024-21626)</p> <p><b>Security Bulletin</b></p> <p><b>Summary</b></p> <p>IBM Cloud Kubernetes Service is affected by a security vulnerability found in the runc component shipped with containerd where an attacker could gain unauthorized access to the host filesystem (CVE-2024-21626).</p> <p><a href="https://www.ibm.com/support/pages/security-bulletin-ibm-cloud-kubernetes-service-affected-containerd-security-vulnerability-cve-2024-21626">https://www.ibm.com/support/pages/security-bulletin-ibm-cloud-kubernetes-service-affected-containerd-security-vulnerability-cve-2024-21626</a></p>

Claim 31	Accused Instrumentalities
	<p data-bbox="709 248 1066 289"><b>containerd Adopters</b></p> <hr data-bbox="709 300 1858 303"/> <p data-bbox="709 336 1417 363">A non-exhaustive list of containerd adopters is provided below.</p> <p data-bbox="709 399 1831 542"><i><b>Docker/Moby engine</b></i> - Containerd began life prior to its CNCF adoption as a lower-layer runtime manager for <code>runc</code> processes below the Docker engine. Continuing today, containerd has extremely broad production usage as a component of the <a href="#">Docker engine</a> stack. Note that this includes any use of the open source <a href="#">Moby engine project</a>; including the Balena project listed below.</p> <p data-bbox="709 578 1772 639"><i><b>IBM Cloud Kubernetes Service (IKS)</b></i> - offers containerd as the CRI runtime for v1.11 and higher versions.</p> <p data-bbox="709 675 1810 781"><i><b>IBM Cloud Private (ICP)</b></i> - IBM's on-premises cloud offering has containerd as a "tech preview" CRI runtime for the Kubernetes offered within this product for the past two releases, and plans to fully migrate to containerd in a future release.</p> <p data-bbox="693 810 1612 837"><a href="https://github.com/moby/containerd/blob/docker/20.10/ADOPTERS.md">https://github.com/moby/containerd/blob/docker/20.10/ADOPTERS.md</a></p>

Claim 31	Accused Instrumentalities
	<h2 data-bbox="741 256 1436 321">Container Runtimes</h2> <div data-bbox="741 375 1738 553"><p><b>Note:</b> Dockershim has been removed from the Kubernetes project as of release 1.24. Read the <a href="#">Dockershim Removal FAQ</a> for further details.</p></div> <p data-bbox="741 621 1722 751">You need to install a <u>container runtime</u> into each node in the cluster so that Pods can run there. This page outlines what is involved and describes related tasks for setting up nodes.</p> <p data-bbox="741 792 1722 878">Kubernetes 1.30 requires that you use a runtime that conforms with the <u>Container Runtime Interface (CRI)</u>.</p> <p data-bbox="741 919 1404 954">See <a href="#">CRI version support</a> for more information.</p> <p data-bbox="693 976 1673 1011"><a href="https://kubernetes.io/docs/setup/production-environment/container-runtimes/">https://kubernetes.io/docs/setup/production-environment/container-runtimes/</a></p>

Claim 31	Accused Instrumentalities
	<h2 data-bbox="716 250 1503 412">Restrict a Container's Syscalls with seccomp</h2> <div data-bbox="747 483 1608 526"> <b>FEATURE STATE:</b> Kubernetes v1.19 [stable]</div> <p data-bbox="716 613 1738 837">Seccomp stands for secure computing mode and has been a feature of the Linux kernel since version 2.6.12. It can be used to sandbox the privileges of a process, restricting the calls it is able to make from userspace into the kernel. Kubernetes lets you automatically apply seccomp profiles loaded onto a <u>node</u> to your Pods and containers.</p> <p data-bbox="716 883 1738 1107">Identifying the privileges required for your workloads can be difficult. In this tutorial, you will go through how to load seccomp profiles into a local Kubernetes cluster, how to apply them to a Pod, and how you can begin to craft profiles that give only the necessary privileges to your container processes.</p> <p data-bbox="690 1185 1373 1218"><a href="https://kubernetes.io/docs/tutorials/security/seccomp/">https://kubernetes.io/docs/tutorials/security/seccomp/</a></p>